dev.nozav.org

Introduction à Subversion

Table des matières

1 Introduction
- 1.1 Objectifs de ce document
- 1.2 Qu'est-ce que `Subversion` ?
- 1.3 Pourquoi `Subversion` ?
- 1.4 Configuration requise

2 Définitions
- 2.1 Notions générales
  - 2.1.1 dépôt (*repository*)
  - 2.1.2 projets
  - 2.1.3 copie de travail (*working copy*)
  - 2.1.4 révisions
- 2.2 Opérations
  - 2.2.1 *checkout*
  - 2.2.2 *import*
  - 2.2.3 *update*
  - 2.2.4 *commit*

3 Utilisation
- 3.1 Création d'un nouveau projet
  - 3.1.1 Projet déjà existant au sein d'un dépôt
  - 3.1.2 Import d'un projet déjà existant en local
- 3.2 Récupération de la dernière version du projet
- 3.3 Mise à jour des modifications dans le dépôt
- 3.4 Récupération d'une version antérieure d'un fichier
  - 3.4.1 Récupération de la dernière version
  - 3.4.2 Récupération d'une version antérieure du dépôt
- 3.5 Gestion des fichiers du dépôt
  - 3.5.1 Ajout d'un fichier
  - 3.5.2 Suppression d'un fichier
  - 3.5.3 Renommer un fichier
- 3.6 Résolution des conflits
- 3.7 Tronc, branches, tags...

4 Outils
- 4.1 Aide intégrée
- 4.2 Informations sur la copie de travail
- 4.3 Voir l'historique des modifications d'un fichier ou projet
- 4.4 Voir le statut de la copie de travail
- 4.5 Parcourir le dépôt
- 4.6 Utiliser les propriétés

5 Clients graphiques
- 5.1 Intérêt d'un client graphique

6 Ressources

7 À propos de ce document

1 Introduction

1.1 Objectifs de ce document

Ce document a pour objectif de servir d'aide-mémoire et de support de formation succinct au logiciel de contrôle de versions `Subversion`[1].

1.2 Qu'est-ce que `Subversion` ?

`Subversion` est un logiciel de gestion de sources et de contrôle de versions. Ce type de programmes a plusieurs fonctions, notamment :

- garder un historique des différentes versions des fichiers d'un projet ;

- permettre le retour à une version antérieure quelconque ;
- garder un historique des modifications avec leur nature, leur date, leur auteur... ;
- permettre un accès souple à ces fichiers, en local ou via un réseau ;
- permettre à des utilisateurs distincts et souvent distants de travailler ensemble sur les mêmes fichiers.

1.3 Pourquoi `Subversion` ?

Il existe un grand nombre de logiciels du même type. Le plus connu d'entre eux et le plus répandu actuellement est sans doute `CVS`[2], mais on peut aussi citer `GNU Arch`, `Bitkeeper`, `Git`, `Superversion`, etc.

Des comparatifs point à point peuvent être trouvés aux adresses suivantes :

http://zooko.com/revision_control_quick_ref.html

http://better-scm.berlios.de/comparison/comparison.html

On pourra justifier rapidement le choix de `Subversion` par les arguments suivants :

- il est multiplateforme ;
- il s'agit d'un logiciel libre ;
- il fonctionne de manière centralisée ;
- son utilisation et son administration sont plus faciles que `CVS` ;
- il supporte plusieurs modes d'accès distants, dont `SSH` et `WebDAV` via `Apache`.

1.4 Configuration requise

Aucune configuration particulière n'est nécessaire pour utiliser `Subversion`, il suffit juste de se procurer un client qui permette de se connecter et de communiquer avec le dépôt. A noter qu'il est très facile d'installer soi-même un serveur `Subversion` en local sur sa machine si l'on souhaite l'utiliser de manière « privée ».
Parmi les clients, on peut citer :

- la ligne de commande, un peu rustique mais qui permet de tout faire[3] ;
- TortoiseSVN (`http://tortoisesvn.tigris.org/`) (Windows)
- RapidSVN (`http://rapidsvn.tigris.org/`), eSvn (`http://esvn.umputun.com/`) (multiplateforme)
- JSVN (`http://jsvn.alternatecomputing.com/`) (java, multiplateforme)
- Subclipse (`http://subclipse.tigris.org/`) (*plugin* pour `Eclipse`)
- AnkhSVN (`http://ankhsvn.tigris.org/`) (*plugin* pour `Visual Studio .Net`)
- psvn.el (`http://xsteve.nit.at/prg/vc_svn/`) (mode pour `Emacs`)
- etc, etc.

Une liste complète de clients est disponible à l'adresse :

http://subversion.tigris.org/links.html#clients

Dans ce qui suit, nous illustrerons les points abordés à l'aide de la ligne de commande, mais les concepts sont les mêmes pour les différents clients.

2 Définitions

2.1 Notions générales

2.1.1 dépôt (*repository*)

Un dépôt `Subversion` est l'emplacement central où sont stockées toutes les données relatives aux projets gérés. Le dépôt est accédé *via* une URL locale ou distante.

Le dépôt contient l'historique des versions des fichiers stockés, les logs enregistrés lors des modifications, les dates et auteurs de ces modifications, etc.

Un dépôt apparaît de l'extérieur comme un système de fichiers composé de répertoires au sein desquels on peut naviguer, lire et écrire selon les permissions accordées.

2.1.2 projets

Au sein d'un dépôt se trouvent un ou plusieurs projets. À chaque projet correspond en général un répertoire situé à la racine du dépôt et qui contient lui-même les fichiers et dossiers du projet proprement dit.
Exemple d'arborescence :

```
(dépôt)--+--/batchxsl--+--/trunk
         |             |
         |             +--/branches
         |             |
         |             +--/tags
         |
         +--/css-------+--/trunk
         |             |
         |             +--/branches
         |             |
         |             +--/tags
         |
         +--/test------+--/rep1
                       |
                       +--/rep2
```

2.1.3 copie de travail (*working copy*)

La copie de travail est un répertoire situé en local sur le poste de l'utilisateur et qui contient une copie d'une révision donnée des fichiers du dépôt. C'est cette copie qui sert de base de travail et qui est modifiée en local avant d'être importée (sauvegardée) vers le dépôt.

2.1.4 révisions

Chaque modification faite au dépôt constitue une révision. Le numéro de révision commence à 1 et augmente de 1 à chaque opération. Sa valeur n'a aucune importance, mais c'est un indicateur qui permet de revenir à une version donnée d'un ou plusieurs fichiers.

2.2 Opérations

2.2.1 *checkout*

Le *checkout* est l'opération qui consiste à récupérer pour la première fois les fichiers déjà existant au sein d'un projet du dépôt. Cette opération ne se fait en général qu'une fois par projet.

Le résultat est une copie de travail.

2.2.2 *import*

L'*import* est l'opération inverse du *checkout*. Elle consiste à placer dans le dépôt des fichiers locaux déjà existants pour y créer un nouveau projet. Cette opération ne se fait en général qu'une fois par projet.

2.2.3 *update*

L'*update* consiste à synchroniser la copie de travail locale avec le dépôt en récupérant la dernière version des fichiers du dépôt.

C'est à cette occasion que des conflits de version peuvent apparaître.

2.2.4 *commit*

Un *commit* est l'opération inverse d'un *update*. Elle consiste à mettre à jour le dépôt à partir de la copie de travail locale. Une nouvelle révision est alors créée. Un log (simple message texte contenant une description des modifications effectuées) doit être saisi à cette occasion.
À noter que pour qu'un *commit* soit possible, il faut que la copie de travail corresponde à la dernière version du dépôt (modifications locales exceptées). Si ce n'est pas le cas, il est nécessaire d'effectuer d'abord un *update* et de résoudre les conflits éventuels avant de réessayer le *commit*.

3 Utilisation

L'utilisation de `Subversion` suit en général un cycle assez répétitif.

3.1 Création d'un nouveau projet

La première chose à faire lors de la première utilisation est de créer un nouveau projet. Deux cas de figure peuvent se présenter : ou bien le projet existe déjà au sein d'un dépôt et il s'agit de récupérer ce projet en local pour en faire une copie de travail, ou bien ce projet existe en local et doit être importé au sein du dépôt.

3.1.1 Projet déjà existant au sein d'un dépôt

Si le projet existe déjà au sein du dépôt, une seule commande suffit pour effectuer un *checkout* et récupérer la dernière version des fichiers : il s'agit de la commande `svn co`.

```
$ svn co https://cens-srv-dev1.ens-lsh.fr/svnrep/formationsvn .
A  credits.htm
A  index.htm
Checked out revision 36.
```

La commande précédente a effectué un *checkout* du projet `formationsvn` (situé dans un répertoire racine du même nom) dans le répertoire courant.

Le résultat de la commande indique que deux fichiers ont été récupérés et que la dernière révision est la révision numéro 36.

3.1.2 Import d'un projet déjà existant en local

Si le projet n'existe pas dans le dépôt et qu'il faut le créer à partir de fichiers locaux, la commande à utiliser est `svn import`. Cette opération n'est en théorie effectuée que par la personne chargée de l'administration du dépôt. Voir la documentation officielle pour plus d'informations.

3.2 Récupération de la dernière version du projet

Avant de travailler sur les fichiers du projet, il faut s'assurer que l'on est bien synchronisé avec le dépôt, c'est à dire que la copie de travail correspond bien à la dernière révision en cours. Pour cela, il faut effectuer un *update* à l'aide de la commande `svn update` :

```
$ svn update
U  index.htm
Updated to revision 37.
```

La commande indique qu'un fichier du répertoire, vraisemblablement modifié par quelqu'un d'autre depuis notre

dernier *update*, a été mis à jour dans notre copie de travail.

3.3 Mise à jour des modifications dans le dépôt

Une fois qu'on a modifié des fichiers, il faut basculer ces modifications au sein du dépôt pour qu'elles soient accessibles aux autres utilisateurs. Cette opération s'effectue à l'aide de la commande `svn commit` :

```
$ svn commit -m "Ajout d'une personne dans les credits"
Sending        credits.htm
Transmitting file data .
Committed revision 38.
```

Toute opération de *commit* s'effectue en indiquant un message décrivant les modifications effectuées (ici directement dans la ligne de commande). Il est possible d'effectuer cette opération sur un répertoire entier, ou sur seulement un ou plusieurs fichiers.

Si des modifications ont eu lieu par un autre utilisateur du dépôt depuis le dernier *update*, un message d'erreur le signale. Il faut alors effectuer un nouvel *update* et résoudre d'éventuels conflits avant de relancer le *commit*.

3.4 Récupération d'une version antérieure d'un fichier

3.4.1 Récupération de la dernière version

Lorsqu'on travaille sur un fichier, il peut arriver que les modifications effectuées ne soient pas bonnes et qu'on souhaite retourner au fichier tel qu'il était lors du dernier *update*. La commande `svn revert` est faite pour ça :

```
$ svn revert credits.htm
Reverted 'credits.htm'
```

Cette commande annule les modifications effectuées depuis le dernier *update*. À noter que tout est effectué en local, et qu'un accès au dépôt n'est pas nécessaire.

3.4.2 Récupération d'une version antérieure du dépôt

On peut aussi souhaiter revenir à une version antérieure d'un fichier situé dans le dépôt. Il faut alors utiliser `svn update` en précisant le numéro de la révision et le ou les fichiers :

```
$ svn update -r 36 credits.htm
U  credits.htm
Updated to revision 36.
```

3.5 Gestion des fichiers du dépôt

`Subversion` propose un ensemble de commandes pour ajouter, supprimer ou renommer des fichiers du dépôt.

3.5.1 Ajout d'un fichier

Il faut utiliser `svn add`. A noter que l'ajout n'est effectif qu'au prochain *commit* :

```
$ svn add liens.htm
A         liens.htm

$ svn commit -m "Ajout du fichier de liens"
Adding         liens.htm
Transmitting file data .
Committed revision 39.
```

3.5.2 Suppression d'un fichier

Il faut utiliser `svn delete` :

```
$ svn delete liens.htm
D         liens.htm

$ svn commit -m "Suppression d'un fichier"
Deleting       liens.htm
Committed revision 40.
```

Là aussi, la suppression n'est effective qu'au *commit* suivant.

3.5.3 Renommer un fichier

Il faut utiliser `svn move` :

```
$ svn move credits.htm merci.htm
A         merci.htm
D         credits.htm

$ svn commit -m "Renommage d'un fichier"
Deleting       credits.htm
Adding         merci.htm
Committed revision 41.
```

3.6 Résolution des conflits

Les conflits peuvent intervenir au moment d'un *update*, lorsque des modifications ont été faites à la fois dans la copie de travail et dans le dépôt. Par exemple, si vous éditez en local un fichier pour lui rajouter une ligne, et qu'un autre utilisateur du dépôt a « commité » entre temps une modification différente sur le même fichier, votre *commit* va générer l'erreur suivante :

```
$ svn commit
Sending        merci.htm
svn: Commit failed (details follow):
svn: Your file or directory 'merci.htm' is probably out-of-date
svn:
The version resource does not correspond to the resource within the
transaction.  Either the requested version resource is out of date
(needs to be updated), or the requested version resource is newer than
the transaction root (restart the commit).
```

Il vous faut alors effectuer un *update*, ce qui va mettre en concurrence les deux versions du ou des fichiers concernés. Deux cas de figure peuvent alors se présenter.

Dans le premier cas, le conflit peut être résolu automatiquement par `Subversion` car les modifications ne concernent pas les mêmes parties du fichier. Dans ce cas vous obtiendrez le message suivant :

```
$ svn update
G  merci.htm
Updated to revision 42.
```

Il est quand même conseillé de vérifier manuellement le résultat de cette résolution « automatique ».

Dans le deuxième cas, les modifications ne peuvent être fusionnées automatiquement car elles concernent les mêmes parties d'un fichier. Dans ce cas un conflit est signalé lors de l'*update* :

```
$ svn update
C  merci.htm
Updated to revision 43.
```

Dans ce cas, deux nouveaux fichiers font leur apparition dans votre copie de travail. Dans l'exemple précédent, on se retrouve avec :

- `merci.htm.mine` : copie du fichier tel qu'il se trouvait dans votre copie de travail, en local, avant de faire l'*update*. C'est la version que vous souhaitiez « commiter » avant de détecter un conflit ;
- `merci.htm.r42` : version du fichier pour la révision 42, c'est à dire lors de votre dernier *update*. C'est la version qui a servi de base pour les deux utilisateurs du dépôt qui ont travaillé en parallèle ;
- `merci.htm.r43` : version du fichier pour la revision 43, c'est à dire la version actuellement dans le dépôt. Il

s'agit de la version modifiée par un autre utilisateur, « commitée » avant votre *update*, et dont le contenu est à l'origine du conflit.

- `merci.htm` : il s'agit d'une version qui, en quelque sorte « résume » les trois autres en faisant apparaître les différences entre versions au sein d'un seul fichier.

Dès lors, le travail consiste à éditer le fichier `merci.htm` jusqu'à ce que le conflit soit résolu[4]. Une fois ce travail terminé, on signale que le conflit est résolu à l'aide de la commande `svn resolved` :

```
$ svn resolved merci.htm
Resolved conflicted state of 'merci.htm'
```

On peut alors effectuer le *commit* final.

3.7 Tronc, branches, tags...

Les notions de tronc, de branches et de *tags* sont assez spécifiques aux logiciels de contrôle de versions. C'est ce qui explique que les arborescences des répertoires de projet contiennent souvent comme premier niveau de sous-répertoires les dossiers `trunk`, `branches` et `tags`.
En général, on définit par « tronc » la version centrale du programme, le développement principal « officiel ».
Une « branche » est en général créée lorsqu'un développement « secondaire » est mis en route, que ce soit pour ajouter une nouvelle fonctionnalité ou parce que certains développeurs souhaitent essayer de prendre une autre direction pour certains aspects du développement. Une branche peut, au bout d'un certain temps, soit être à nouveau fusionnée dans le « tronc », soit disparaître, soit donner lieu à un nouveau programme.
La notion de *tags* correspond en partie à celle de *release*, c'est à dire de marquage d'une certaine révision du projet comme composant une version du projet. Une fois que le développement a atteint une certaine stabilité, on pourra par exemple créer un *tag* pour marquer la sortie de la version 1.0. Ceci permettra de revenir facilement à cette version, indépendamment du numéro de révision sous-jacent correspondant.
Nous n'entrerons pas dans le détail de ces concepts et commandes ici, mais on peut juste citer que la création de branches ou de *tags* ne sont en fait que des copies créées par la commande `svn copy`. La commande `svn switch`, elle, permet de faire passer la copie de travail d'une branche à une autre.

4 Outils

4.1 Aide intégrée

Une aide est intégrée à l'interface en ligne de commande. Pour obtenir de l'aide générale, et notamment la liste des commandes possibles, il suffit de faire :

```
svn help
```

Pour obtenir de l'aide sur une commande particulière, utiliser :

```
svn help <nom de la commande>
```

4.2 Informations sur la copie de travail

Pour obtenir des informations sur la copie de travail en cours, on peut utiliser `svn info` :

```
$ svn info
Path: .
URL: https://cens-srv-dev1.ens-lsh.fr/svnrep/formationsvn
Repository UUID: 090fa6ab-88f7-0310-b83e-cd111ae4905a
Revision: 40
Node Kind: directory
Schedule: normal
Last Changed Author: jbarnier
Last Changed Rev: 40
Last Changed Date: 2005-05-30 14:58:11 +0200 (lun, 30 mai 2005)
```

4.3 Voir l'historique des modifications d'un fichier ou projet

La commande `svn log` permet d'afficher l'historique de toutes les modifications d'un fichier donné en paramètre ou d'un projet entier :

```
$ svn log index.htm
 ------------------------------------------------------------------------
 r37 | jbarnier | 2005-05-30 14:42:26 +0200 (lun, 30 mai 2005) | 2 lines

 Modification du titre de la page

 ------------------------------------------------------------------------
 r36 | jbarnier | 2005-05-30 14:34:05 +0200 (lun, 30 mai 2005) | 2 lines

 Ajout de deux fichiers de test.

 ------------------------------------------------------------------------
```

La commande `svn blame` permet d'obtenir des informations sur un fichier ligne par ligne, avec la révision et l'auteur correspondants :

```
$ svn blame index.htm
    36   jbarnier <html>
    36   jbarnier <head></head>
    36   jbarnier <body>
    37   jbarnier <h1>Un bien beau titre, vraiment</h1>
    36   jbarnier </body>
    36   jbarnier </html>
```

4.4 Voir le statut de la copie de travail

La commande `svn status` permet d'avoir des informations sur l'état de la copie de travail depuis le dernier *update* :

```
$ svn status -v
?                                       credits.htm
               40       40 jbarnier     .
               41       41 jbarnier     merci.htm
               40       37 jbarnier     index.htm
```

4.5 Parcourir le dépôt

La commande `svn list` permet d'afficher le contenu du dépôt à distance :

```
$ svn list https://cens-srv-dev1.ens-lsh.fr/svnrep/formationsvn
index.htm
merci.htm
```

4.6 Utiliser les propriétés

Les propriétés sont des attributs attachés à un ou plusieurs fichiers du dépôt et qui permettent des comportements particuliers.
Sans entrer dans le détail, on ne citera que deux types de propriétés. Tout d'abord, la propriété `svn:ignore` permet de retirer explicitement certains fichiers du contrôle de version. Par exemple, si votre éditeur de texte enregistre systématiquement une copie de sauvegarde de vos fichiers avec l'extension `.bak`, il peut être intéressant de systématiquement mettre de côté ces fichiers en positionnant la propriété correspondante :

```
$ svn propset svn:ignore *.bak .
property 'svn:ignore' set on '.'
```

Une autre utilisation intéressante concerne l'utilisation de mots-clés. Ceux-ci sont des identifiants insérés dans les fichiers du projet et qui seront remplacés au moment du *commit* par des informations propres à `Subversion`, comme le nom du fichier, le numéro de révision, l'auteur et la date de la dernière modification, etc.
Par exemple, si vous insérez la chaîne `$Id$` dans votre fichier, celle-ci sera automatiquement remplacée par un résumé de ces informations. Voici la valeur correspondant au fichier source de ce document :

```
$Id: formation_svn.tex 135 2005-08-02 09:47:51Z julien $
```

Pour que ces mots-clés fonctionnent, il faut positionner la propriété `svn:keywords` de manière adéquate pour les fichiers concernés :

```
$ svn propset svn:keywords "Id" index.htm
property 'svn:keywords' set on 'index.htm'
```

5 Clients graphiques

5.1 Intérêt d'un client graphique

Les clients graphiques ne permettent pas de faire plus, mais proposent des interfaces plus élaborées que la ligne de commande. Ils permettent notamment de naviguer dans le dépôt comme dans un explorateur de fichiers, d'afficher les informations de manière plus structurée, de garder un historique des logs saisis, etc.
`TortoiseSVN`, par exemple, est un client particulièrement bien intégré à Windows, puisqu'il s'ajoute directement au menu contextuel de l'explorateur de fichiers. Les commandes sont les mêmes que celles décrites dans ce document, mais elles sont lancées par un menu et bénéficient d'interfaces graphiques plus conviviales.

6 Ressources

- Site officiel de Subversion :
  http://subversion.tigris.org/
- FAQ de Subversion :
  http://subversion.tigris.org/faq.html
- Documentation complète (en anglais) :
  http://svnbook.red-bean.com/
- Un tutoriel en français :
  http://toutprogrammer.com/print_19.html
- Documentation de TortoiseSVN (en anglais) :
  http://tortoisesvn.tigris.org/docs.html

7 À propos de ce document

Ce document est publié sous licence *Creative Commons Attribution*. Vous pouvez voir une copie de cette licence à l'adresse http://creativecommons.org/licenses/by/2.5/.
Copyright © 2005 Julien Barnier.
Pour tout commentaire ou suggestion, n'hésitez pas à m'écrire à l'adresse julien (at) nozav.org.

---

1 http://subversion.tigris.org/

2 http://www.cvshome.org/. `Subversion` est souvent cité comme le successeur « officiel » de `CVS`.

3 Pour utiliser la ligne de commande, il faut télécharger et installer `Subversion` depuis le site officiel.

4 Les clients graphiques proposent parfois des outils plus conviviaux pour résoudre « visuellement » les conflits de version. `TortoiseSVN` utilise le programme associé `TortoiseMerge`.

Ce document a été traduit de LaTeX par HeVeA

Dernière modification : 2009-01-10 11:21 *Généré par GNU Emacs et Muse mode.*